PSis - Programação de Sistemas 2013/2014
Exame 1ª Época, 9 de Junho de 2014, 8h00, Duração: 2h

1. [1 valor] Quantos processos são criados pelo seguinte código? Justifique.

```
for(i=0; i < 2; i++){
          if(fork()!=0){
               printf("pai\n");
          }else{
               printf("filho\n");
          }
}
```

São criados 3 processos: o pai cria dois filhos (i = 0, i=1). O primeiro filho a ser criado herda todo o estado do pai estando dentro do ciclo, pelo que cria um neto (quando i = 1).

2. [2 valores] Indique 3 dos mecanismos do UNIX que permitem a transferência de informação entre processos. Para cada mecanismo indique:

- como se processa a transferência de informação;
- que tipo informação/dados podem ser transferidos;
- que tipo de processos intervêm na comunicação;
- direcção da comunicação.

Exclua nesta resposta pipes, FIFOs ou sockets.

Estado partilhado entre o pai e o filho
Sinais
Retorno/wait
Variáveis de ambiente
Ficheiros

3. [1 valor] A divisão de um problema em várias *threads* ou processos permite um melhor aproveitamento dos recursos computacionais.
Indique duas vantagens da utilização de *threads* em detrimento do uso de processos, aquando da paralelização de um problema.

Tempo de inicializaçcão menor
Memória partilhada
Menos recursos ocupados

4. [1 valor] Actualmente diversos processadores fornecem para um mesmo *core* (núcleo) a possibilidade de executar concorrentemente duas *threads* (SMT/Hyperthreading), fornecendo ao SO dois *cores* virtuais.
Indique quais as alterações arquitecturais (entre um *core* com e sem SMT/Hyperthreading) que permitem tal execução e descreva quais os tipos de instruções/aplicações que permitem um aumento de desempenho aquando do uso simultâneo de dois *cores* virtuais.

O Hyperthreading utiliza unidades aritméticas de reais e inteiros distintas para conseguir paralelismo, no entanto necessita que o processador tenha todos os registos duplicados.
Só se consegue tirar partido to Hyperthreading, se as uma das threads/processos executar muitas operaçcões sobre inteiros e a outra operaçcões sobre reais.

5. [1 valor] Descreva de que modo é possível usando pipes estabelecer um canal de comunicação

bidireccional entre dois processos.
Escreva o código em C que o implementa. No mesmo programa, e após a criação do canal, exemplifique o envio de dois inteiros (um em cada sentido) entre os processos.

Os pipes são unidirecionais, pelo que será necessário usar dois pipes, para que se consiga implementar um canal bidirecional entre dois processos:

```
int main(){
  int pai_filho[2];
  int filho_pai[2];
  int n;

  pipe(pai_filho);
  pipe(filho_pai);
  if(fork()== 0){
       // filho
       printf("filho\n");
       close(filho_pai[0]);
       close(pai_filho[1]);

       n = 11;
       write(filho_pai[1], &n, sizeof(n));
       read(pai_filho[0], &n, sizeof(n));
       printf("filho leu %d\n", n);
  }else{
       // pai
       printf("filho\n");
       close(filho_pai[1]);
       close(pai_filho[0]);

       n = 10;
       write(pai_filho[1], &n, sizeof(n));
       read(filho_pai[0], &n, sizeof(n));
       printf("pai leu %d\n", n);
  }
}
```

6. [1 valor] Descreva duas diferenças entre um FIFO (ou named pipe) e um socket do domínio UNIX.

FIFO	Socket UNIX
Unidirecional Stream novo canal explícito emissor/recpetor anónimos comunicação “um para muitos”	Bidirecional mensagens novo socket automático (accept) identificação do emissor/receptor comunicaçcão ponto a ponto

PSis - Programação de Sistemas 2013/2014
Exame 1ª Época, 9 de Junho de 2014, 8h00, Duração: 2h

7. [3 valores] Pretende-se construir uma aplicação em que processos Produtor escrevem mensagens que são lidas por processos Consumidor. As caraterísticas do sistema são as seguintes:

- As mensagens são colocadas em 4 filas, cada uma associada a um nível de prioridade. As mensagens são todas do mesmo tamanho. As filas de mensagens têm espaço para NMSG mensagens, em que NMSG é uma constante.
- Quando um Produtor pretende escrever uma mensagem numa fila e a fila está cheia (i.e., já tem NMSG ainda não consumidas), fica bloqueado.
- O processo Consumidor retira uma mensagem da fila mais prioritária, de entre as filas que têm mensagens. Se não houver mensagens em nenhuma das filas deve ficar bloqueado.

Programe em pseudo-código as rotinas que permitem inserir e retirar mensagens das ditas filas, com a seguinte sintaxe:

- void inserir(int prioridade, struct mensagem m);
- struct mensagem retirar();

Utilize semáforos e mutexes para sincronizar os processos, com as usuais funções de esperar e assinalar. Estas funções recebem um único parâmetro, o semáforo ou o mutex.

Considere as seguintes estruturas:

- struct mensagem – estrutura que representa as mensagens.
- struct mensagem[NIVEL][NMSG] buffer – matriz de mensagens onde buffer[NIVEL] representa a fila de mensagens correspondente à prioridade associada ao NIVEL. Considere que o 1º índice de uma tabela é o 0.

```
define NIVEL 4
define NMSG 3

sem_t sem_posOcupadas = 0; /*indica de há posições
ocupadas, em algum nível, qq q seja*/
sem_t sem_posLivres[NIVEL] = {3,3,3,3}; /*indica se há
posições livres num determinado nível*/
mutex_t mutex = 1;
int indiceLeitra[NIVEL] = {0,0,0,0};
int indiceEscrita[NIVEL] = {0,0,0,0};
int posOcupadas[NIVEL] = {0,0,0,0};

void inserir(int prioridade, struct mensagem m){
     esperar(sem_posLivres[prioridade]);
     esperar(mutex);

     buffer[prioridade][indiceEscrita[prioridade]]=m;
     indiceEscrita[prioridade]++;
     indiceEscrita[prioridade]%=NMSG;
     posOcupadas[prioridade]++;

     assinalar(mutex);
```

```
        assinalar(sem_posOcupadas);
}

struct mensagem retirar(){
        int prioridade = -1, i;
        struct mensagem res;

        esperar(sem_posOcupadas);
        esperar(mutex);

        for (i=0; i<NIVEL && prioridade==-1; i++)
                if (posOcupadas[i]>0) prioridade = i;
        res = buffer[prioridade]
[indiceLeitura[prioridade]];
        indiceLeitura[prioridade]++;
        indiceLeitura[prioridade]%=NMSG;
        posOcupadas[prioridade]--;

        assinalar(mutex);
        assinalar(sem_posLivres[prioridade]);
        return res;
}
```